1. Pile inversée

(a) Si on veut utiliser les primitives de gestion de pile, on a besoin d'une <u>seconde pile</u>.

```
inverse_pile (pile p) {
    newp = new pile ;
    while (! p.vide())
        newp.empile (p.depile())
    return newp ;
}
```

(b) Si on a accès à l'implémentation de la pile, on peut l'inverser <u>dans le tableau lui-même</u>:

```
inverse_pile (pile p) {
    haut = p.s
    bas = p.1
    while (haut > bas) {
        echange (p.tab, haut, bas);
        bas ++ ;
        haut -- ;
    }
}
```

Hypothèses p.tab est le tableau dont le sommet est dans la case p.s.

(c) Complexité :

Nb d'opérations : n pour (a)
$n/2$ pour (b)

Nb de transferts : n pour (a)
$3n/2$ pour (b)
(Un échange correspond à 3 transferts).

Nb de cases mémoire : $2n$ pour (a)
n pour (b)

2. Arbre dictionnaire.

(a)
```
avant (X, Y) {
    if domine (X, Y) return True ;
    if domine (Y, X) return false ;
    // X & Y ne sont pas sur la même branche
    Z = premier_ancetre_commun (X, Y) ;
    return el_chemin (Z, X, 1) < el_chem (Z, Y, 1) ;
}
```

(b)
```
domine (X, Y) {
    n = len (liste_fils (X)) ;
    for ( i=1 ; i≤n ; i++ ) {
        z = element ( liste_fils (X), i)
        if (z == X) return True ;
        if domine (z, X) return True ;
    }
    return false ;
}
```

(c) fils (X) : chaîne X + caractère 'a'

existe_frere (X) : la dernière lettre de X n'est pas 'z'

faire (X) : remplacer la dernière lettre de X par la suivante dans l'alphabet

père (X) : chaîne X privée du dernier caractère

profondeur (X) : longueur de la chaîne X

(suite n° 2)

(d)

```
premier_ancetre_commun (X, Y) {
    int i = 1;
    string Z = "";   (chaine vide)
    while (X[i] == Y[i]) {
        i++;
        Z = Z + X[i];
    }
    return Z;
}
```

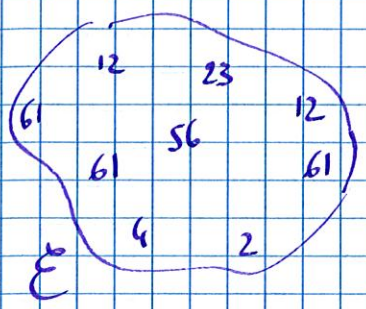
- il s'agit du préfixe commun de X & Y
- la primitive chemin(X, Y) correspond aux lettres de Y qui restent quand on enlève le préfixe X (si X n'est pas préfixe de Y, le chemin() est vide).

(e)

```
avant (X, Y) {
    i = 1
    while ( X[i] == Y[i] )
        i++;
    return X[i] < Y[i];
}
```

3. Multi-ensembles

exemple :



et codé :

1	2	3	4	..	12	...	23	...	56	...	61		100
0	1	0	1	2	0	0	0	1	..	1	0	0	3	.....	0	.....

L'intersection de 2 multi-ensemble est le + gd multi-ensemble qui est inclus dans chacun des 2 :

donc, p.ex, si 61 apparaît 3 fois dans E, et 2 fois dans F, alors il doit figurer 2 fois dans $E \cap F$

d'où le code :

```
intersection (int X[100], int Y[100]) {
    int Z[100];
    for (int i = 1; i < 100; i++)
        Z[i] = min (X[i], Y[i]);
    return Z;
}
```

(où min (A,B) renvoie la plus petite valeur entre A & B).